下载APP

18 | 图像分类（下）：如何构建一个图像分类模型?

2021-11-22 方远

《PyTorch深度学习实战》 课程介绍 >

讲述：方远
时长 14:42 大小 13.47M

你好，我是方远。欢迎来到第 18 节课的学习。

我相信经过上节课的学习，你已经了解了图像分类的原理，还初步认识了一些经典的卷积神经网络。

正所谓“纸上得来终觉浅，绝知此事要躬行”，今天就让我们把上节课的理论知识应用起来，一起从数据的准备、模型训练以及模型评估，从头至尾一起来完成一个完整的图像分类项目实践。

问题回顾

我们先来回顾一下问题背景，我们要解决的问题是，在众多图片中自动识别出极客时间 Logo 的图片。想要实现自动识别，首先需要分析数据集里的图片是啥样子的。

那我们先来看一张包含极客时间 Logo 的图片，如下所示。

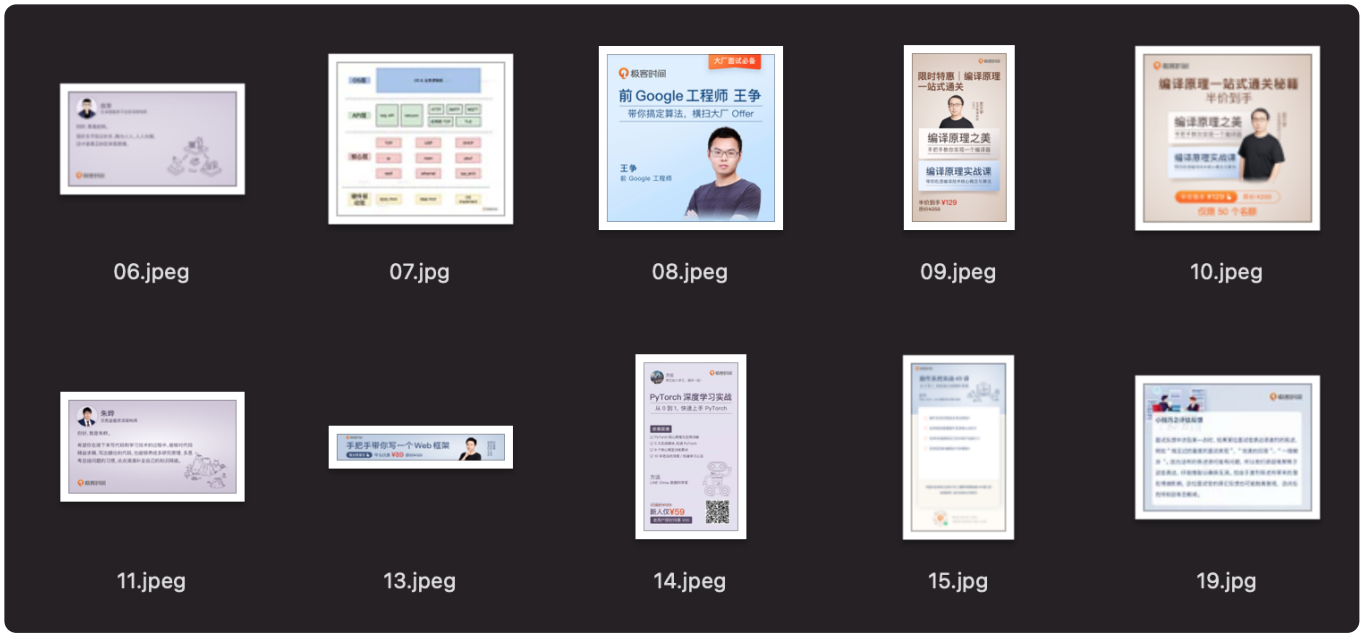
你可以看到，Logo 占整张图片的比例还是比较小的，所以说，如果这个项目是真实存在的，目标检测其实更加合适。不过，我们可以将问题稍微修改一下，修改成自动识别极客时间宣传海报，这其实就很适合图像分类任务了。

数据准备

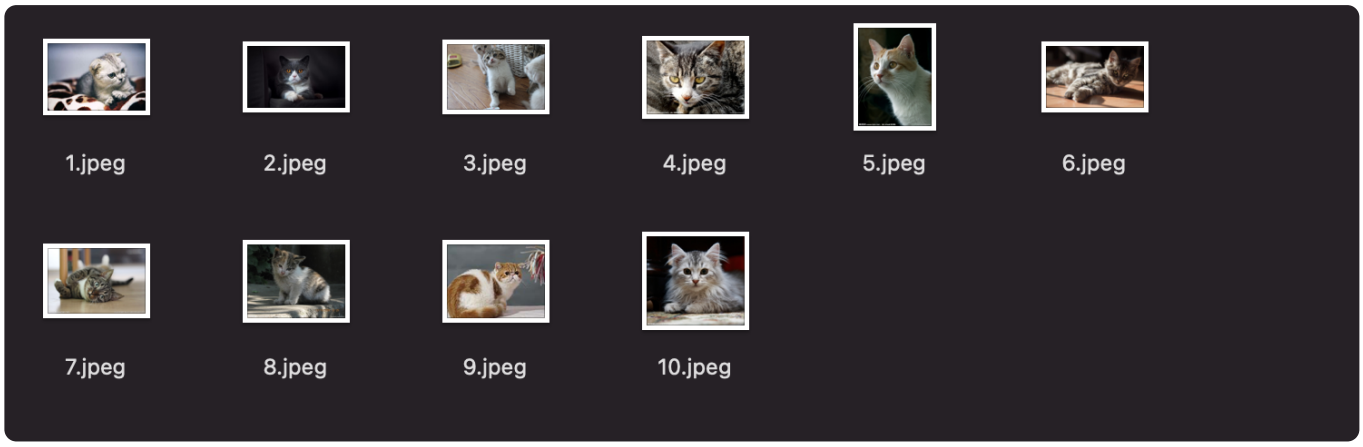
相比目标检测与图像分割来说，图像分类的数据准备还是比较简单的。在图像分类中，我们只需要将每个类别的图片放到指定的文件夹里就行了。

下图是我的图片组织方式，文件夹就是图片所属的类别。

logo 文件夹中存放的是 10 张极客时间海报的图片。



而 others 中，理论上应该是各种其它类型的图片，但这里为了简化问题，我这个文件夹中存放的都是小猫的图片。



模型训练

好啦，数据准备就绪，我们现在进入模型训练阶段。

今天我想向你介绍一个在最近 2 年非常受欢迎的一个网络——EfficientNet。它为我们提供了 B0 ~ B7，一共 8 个不同版本的模型，这 8 个版本有着不同的参数量，在同等参数量的模型中，它的精度都是首屈一指的。因此，这 8 个版本的模型可以解决你的大多数问题。

EfficientNet

我先给你解读一下 EfficientNet的这篇论文，这里我着重分享论文的核心思路还有我的理解，学有余力的同学可以在课后自行阅读原文。

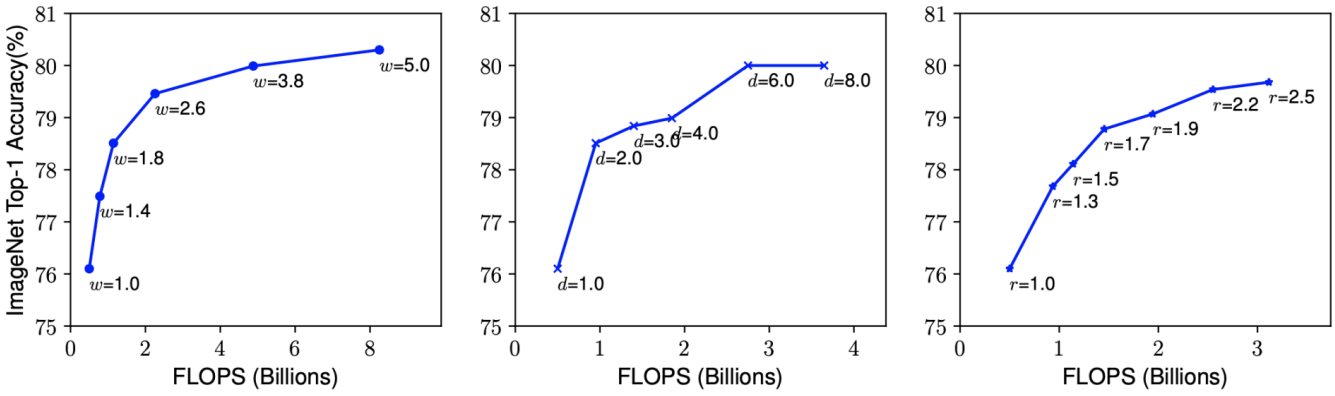
EfficientNet 一共有 B0 到 B7，8 个模型，参数量由少到多，精度也越来越高，具体你可以看看后面的评价指标。

在之前的那些网络，要么从网络的深度出发，要么从网络的宽度出发来优化网络的性能，但从来没有人将这些方向结合在一起考虑。**而 EfficientNet 就做了这样的尝试，它探索了网络深度、网络宽度、图像分辨率之间的最优组合。**

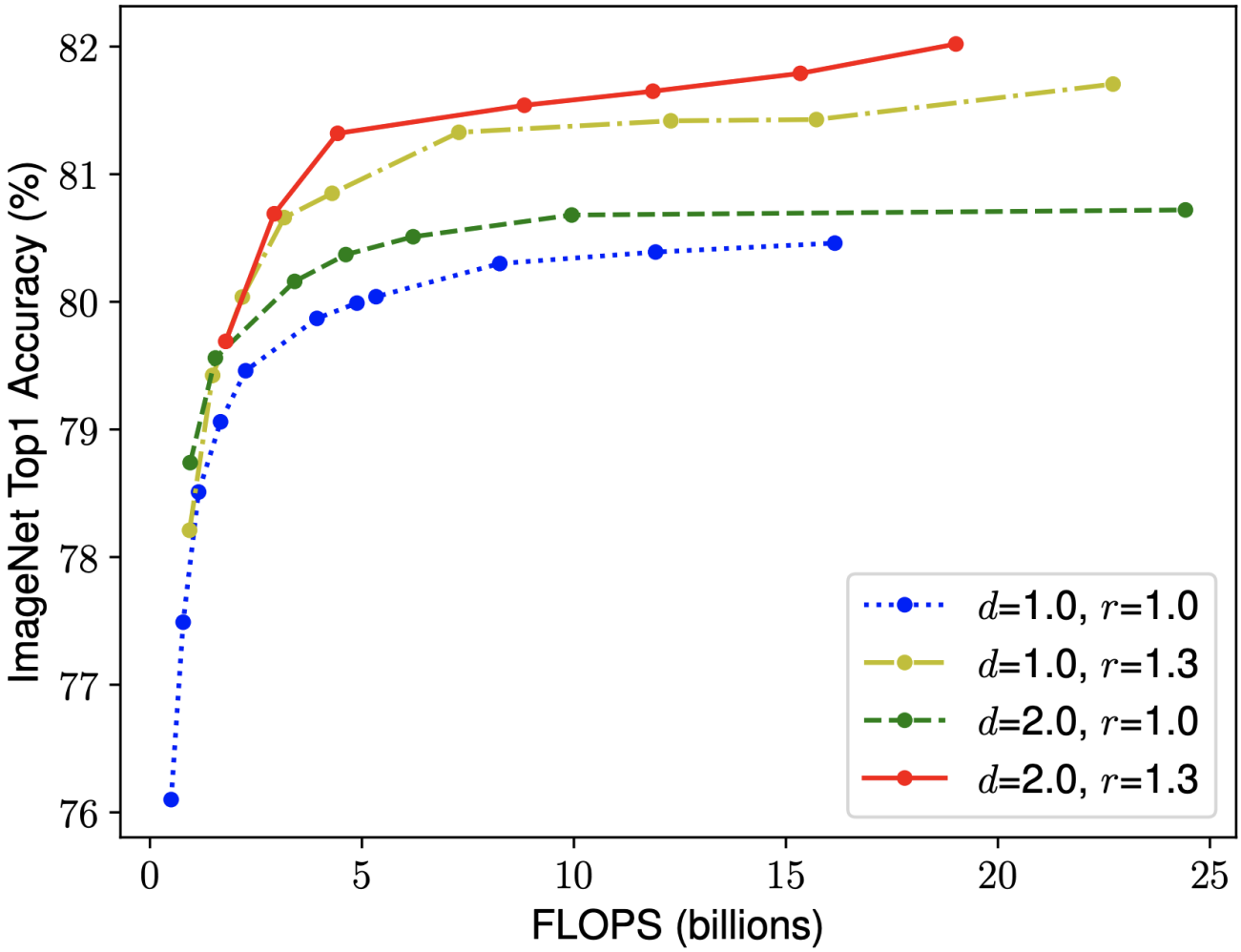
EfficientNet 利用一种复合的缩放手段，对网络的深度 depth、宽度 width 和分辨率 resolution 同时进行缩放（按照一定的缩放规律），来达到精度和运算复杂度 FLOPS 的权衡。

但即使只探索这三个维度，搜索空间仍然很大，所以作者规定只在 B0（作者提出的 EfficientNet 的一个 Baseline）上进行放大。

首先，作者比较了单独放大这三个维度中的任意一个维度效果如何。得出结论是放大网络深度或网络宽度或图像分辨率，均可提升模型精度，但是越放大，精度增加越缓慢，如下图所示：



然后，作者做了第二个实验，尝试在不同的 r（分辨率），d（深度）组合下变动 w（宽度），得到下图：



结论是，得到更高的精度以及效率的关键是平衡网络宽度，网络深度，图像分辨率三个维度的缩放倍率 (d, r, w)。

因此，作者提出了混合维度放大法，该方法使用一个 ϕ（混合稀疏）来决定三个维度的放大倍率。

深度 depth : $d = \alpha^{\phi}$

宽度 width : $w = \beta^{\phi}$

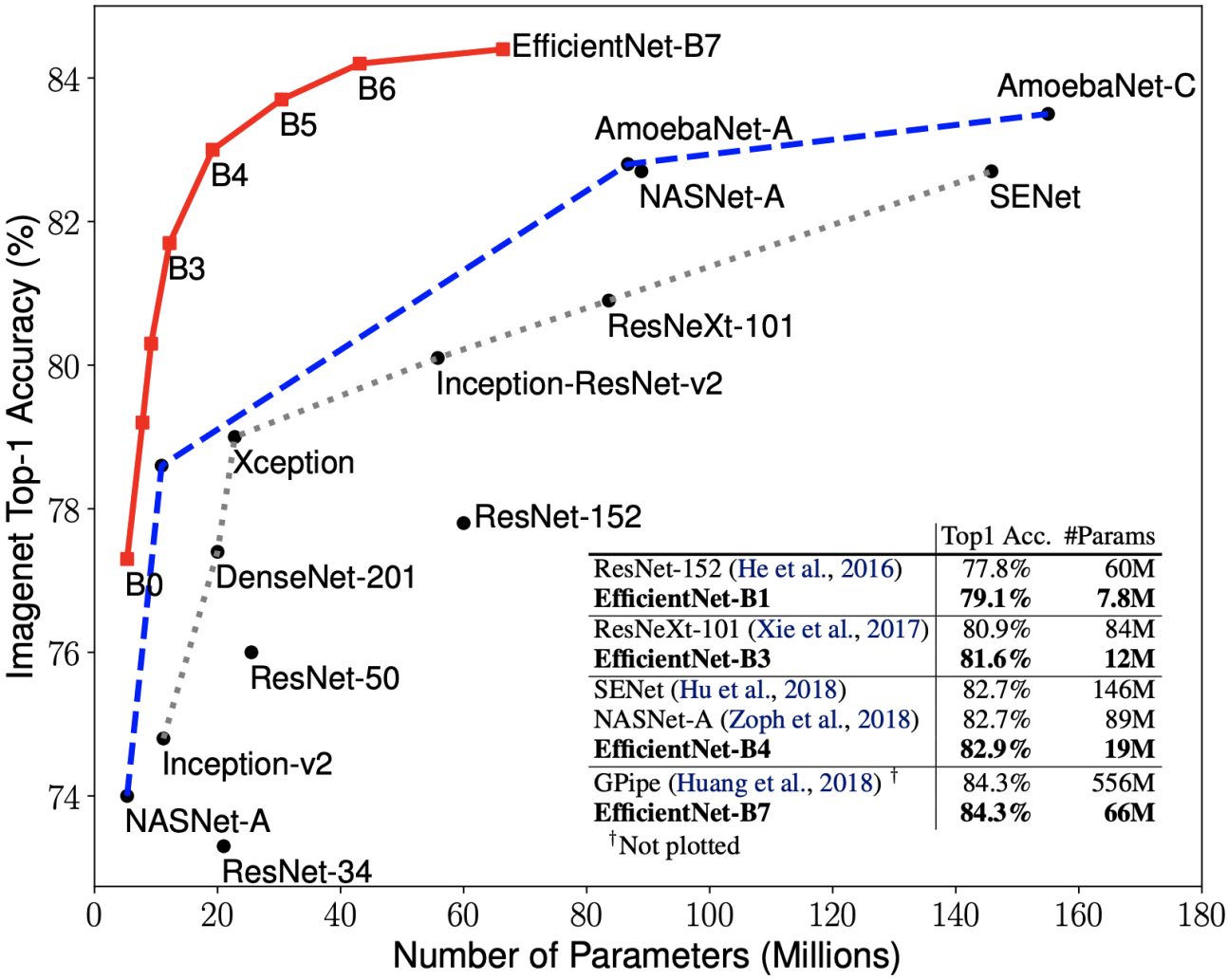
分辨率 resolution: $r = \gamma^{\phi}$

$$s.t.\ \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \quad \alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

第一步，固定 ϕ 为 1，也就是计算量为 2 倍，使用网格搜索，得到了最佳的组合，也就是 $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$。

第二步，固定 $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$，使用不同的混合稀疏 ϕ，得到了 B1~B7。

整体评估效果如下图所示：



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

从评估结果上可以看到，EfficientNet 的各个版本均超过了之前的一些经典卷积神经网络。

EfficientNet v2 也已经被提出来了，有时间的话你可以自己去看看。

我们不妨借助一下 EfficientNet 的 GitHub，它里面有训练 ImageNet 的 demo(demo/imagenet/main.py)，接下来我们一起看看它的核心代码，然后精简一下代码，把它运行起来 (Torchvision 也提供了 EfficientNet 的模型，课后你也可以自己试一试)。

这里我们再回顾一下，之前说的机器学习 3 件套：

1. 数据处理

2. 模型训练（构建模型、损失函数与优化方法）

3. 模型评估

接下来我们就挨个看看这些步骤。你需要先把 https://github.com/lukemelas/EfficientNet-PyTorch给克隆下来，我们只使用 efficientnet_pytorch 中的内容，它包含着模型的网络结构。

之后我们来创建一个叫做 geektime 的项目文件夹，然后把 efficientnet_pytorch 放进去。

在开始之前，我先把程序需要的参数给你列一下，在下面的讲解中，我们就直接使用这些参数了。当你在实现今天代码的时候，需要将这些参数补充到代码中（可以使用 argparsem 模块）。

参数	参数值	说明
train_data	./data/train	训练文件夹的位置
--image-size	224	输入图片的宽与高，B0推荐224
--batch-size	32	batchsize数
--workers	4	Dataloader的worker数
--epochs	20	epoch数
--lr	0.001	学习率
--checkpoint-dir	./checkpoints	模型保存位置
--save-interval	1	保存间隔，每1个epoch保存一次
--momentum	0.9	momentum动量
--weight-decay	1E-04	权重衰减
--arch	efficientnet-b0	使用网络结构
--pretrained	TRUE	是否使用预训练模型

极客时间

好，下面让我们正式开始动手。

加载数据

首先是数据加载的环节，我们创建一个 dataset.py 文件，用来存储与数据有关的内容。dataset.py 如下（我省略了模块的引入）。

复制代码

```
# 作者给出的标准化方法
def _norm_advprop(img):
    return img * 2.0 - 1.0

def build_transform(dest_image_size):
    normalize = transforms.Lambda(_norm_advprop)

    if not isinstance(dest_image_size, tuple):
        dest_image_size = (dest_image_size, dest_image_size)
    else:
        dest_image_size = dest_image_size

    transform = transforms.Compose([
        transforms.RandomResizedCrop(dest_image_size),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize
    ])

    return transform

def build_data_set(dest_image_size, data):
    transform = build_transform(dest_image_size)
    dataset=datasets.ImageFolder(data, transform=transform, target_transform=N

    return dataset

```

这部分代码完成的工作是，通过 build_data_set 构建数据集。这里我们使用了 torchvision.datasets.ImageFolder 来创建 Dataset。ImageFolder 能将按文件夹形式的组织数据生成到一个 Dataset。

在这个例子中，我传入的训练集路径为’./data/train’，你可以看看开篇的截图。

ImageFolder 会自动的将同一文件夹内的数据打上一个标签，也就是说 logo 文件夹的数据，ImageFolder 会认为是来自同一类别，others 文件夹的数据，ImageFolder 会认为

是来自另外一个类别。

我们这个精简版只构建了训练集的 Dataset，当你看 Efficient 官方代码的时候，在验证集的构建过程中，你需要留意一下验证集的 transforms。

我认为，这里这么做是有点问题的，原因是 Resize 中 size 参数如果是个 tuple 类型，则直接按照 size 的尺寸进行 resize。如果是一个 int 的时候，如果图片的 height 大于 width，则按照 (size * height/width, size) 进行 resize。

在作者的原始程序中，imag_size 是个 int，而不是 tuple。所以按照这种先 resize 再 crop 的方式处理一下，对长宽比比较大的图片来说，效果不是很好。

让我们实际验证一下这个想法，我将开篇的例子（也就是那张海报图）的 image_size 设定为 224 后，用上述的方式进行处理后，获得下图。

你看，是不是缺少了很多信息？

所以，如果在我们的例子中使用作者的程序，就需要做一下修改。把这里的代码逻辑修改为如果 image_size 不是 tuple，先将 image_size 转换为 tuple，并且也不需要 crop 了。代码如下所示：

复制代码

```
if not isinstance(image_size, tuple):
    image_size = (image_size, image_size)
else:
    image_size = image_size

transform = transforms.Compose([
```

```
    transforms.Resize(image_size, interpolation=Image.BICUBIC),
    transforms.ToTensor(),
    normalize,
])
```

训练的主程序我们定义在 main.py 中，在 main.py 中的 main() 中，进行数据的加载，如下所示。

然后，我们通过 for 循环一个一个 Epoch 的调用 train 方法进行训练就可以了。

复制代码

```
# 省略了一些模块的引入
from efficientnet import EfficientNet
from dataset import build_data_set

def main():
    # part1: 模型加载 (稍后补充)
    # part2: 损失函数、优化方法(稍后补充)
    train_dataset = build_data_set(args.image_size, args.train_data)

    train_loader = torch.utils.data.DataLoader(
        train_dataset,
        batch_size=args.batch_size,
        shuffle=True,
        num_workers=args.workers,
        )

    for epoch in range(args.epochs):
        # 调用train函数进行训练，稍后补充
        train(train_loader, model, criterion, optimizer, epoch, args)
        # 模型保存
        if epoch % args.save_interval == 0:
            if not os.path.exists(args.checkpoint_dir):
                os.mkdir(args.checkpoint_dir)
            torch.save(model.state_dict(), os.path.join(args.checkpoint_dir,
                    'checkpoint.pth.tar.epoch_%s' % epoch))
```

创建模型

接下来，我们来看看如何创建模型，这一步我们直接使用作者给出的 Efficient 模型。在上面代码注释中的 part1 部分，用下述代码即可加载 EfficientNet 模型。

复制代码

```
    args.classes_num = 2
    if args.pretrained:
        model = EfficientNet.from_pretrained(args.arch, num_classes=args.class
                advprop=args.advprop)
        print("=> using pre-trained model '{}'".format(args.arch))
    else:
        print("=> creating model '{}'".format(args.arch))
        model = EfficientNet.from_name(args.arch, override_params={'num_classe
    # 有GPU的话，加上cuda()
    #mode.cuda()
```

这段代码是说，如果 pretrained model 参数为 True，则自动下载并加载 pretrained model 后进行训练，否则是使用随机数初始化网络。

from_pretrained 与 from_name 中，都需要修改一下 num_classes，将 EfficientNet 的全连接层修改我们项目对应的类别数，这里的 args.classes_num 为 2（logo 类与 others 类）。

模型微调

模型微调在第 8 节课和第 14 节课时说过，这个概念比较重要，我们一起再复习一下。

Pretrained model 一般是在 ImageNet（也有可能是 COCO 或 VOC，都是公开数据集）上训练过的模型，我们可以直接把它在 ImageNet 上训练好的模型参数直接拿过来，在其基础上训练我们自己的模型，这就是模型微调。

所以说，**如果有 Pretrained model，我们一定会使用 Pretrained model 进行训练，收敛速度会快。**

使用 Pretrained model 的时候要注意一点，在 ImageNet 上训练后的全连接层一共有 1000 个节点，所以使用 Pretrained model 的时候只使用全连接层以外的参数。

在上述代码的 EfficientNet.from_pretrained 中，会通调用 load_pretrained_weights 函数，调用之前 num_classes 已经被修改为 2（logo 与 others），所以说传入 load_pretrained_weights 的 load_fc 参数为 False，也就是说不会加载全连接层的参数。load_pretrained_weights 的调用如下所示：

复制代码

```
load_pretrained_weights(model, model_name, load_fc=(num_classes == 1000), advp
```

load_pretrained_weights 函数中包含下面这段代码，就像刚才所说，如果不加载全连接层，则删除 _fc 的 weight 与 bias：

复制代码

```
if load_fc:
    ret = model.load_state_dict(state_dict, strict=False)
    assert not ret.missing_keys, 'Missing keys when loading pretrained weights
else:
    state_dict.pop('_fc.weight')
    state_dict.pop('_fc.bias')
    ret = model.load_state_dict(state_dict, strict=False)
```

设定损失函数与优化方法

最后要做的就是设定损失函数与优化方法了，我们将下面的代码补充到 part2 部分：

复制代码

```
criterion = nn.CrossEntropyLoss() # 有GPU的话加上.cuda()

optimizer = torch.optim.SGD(model.parameters(), args.lr,
                            momentum=args.momentum,
                            weight_decay=args.weight_decay)
```

到这里，我们就完成训练的所有准备了，只要再补充好 train 函数就可以了，代码如下。下面的代码的原理我们在第 13 节课中已经讲过了，记不清的可以去回顾一下。

复制代码

```
def train(train_loader, model, criterion, optimizer, epoch, args):
    # switch to train mode
    model.train()

    for i, (images, target) in enumerate(train_loader):
        # compute output
        output = model(images)
        loss = criterion(output, target)
        print('Epoch ', epoch, loss)

```

```
        # compute gradient and do SGD step
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

不过在我的程序里，保存了若干个 Epoch 的模型，我们应该怎么选择呢？这就要说到模型的评估环节。

模型评估

对于分类模型的评估来说，有很多评价指标，例如准确率、精确率、召回率、F1-Score 等。其中，**我认为最直观、最有说服力的就是精确率与召回率**，这也是我在项目中观察的主要是指标。下面我们依次来看看。

混淆矩阵

在讲解精确率与召回率之前，我们先看看混淆矩阵这个概念。其实精确率与召回率就是通过它计算出来的。下表就是一个混淆矩阵，正例就是 logo 类，负例就是 others 类。

		真实类别	
		正例	负例
预测结果	正例	True Positive (TP)	False Positive (FP)
	负例	False Negative (FN)	True Negative (TN)

极客时间

根据预测结果和真实类别的组合，一共有四种情况：

1.TP 是说真实类别为 Logo，模型也预测为 Logo；

2.FP 是说真实类别为 Others，但模型预测为 Logo；

3.FN 是说真实类别为 Logo，但模型预测为 Others；

4.TN 是说真实类别为 Others，模型也预测为 Others；

精确率的计算方法为：

$$precision = \frac{TP}{(TP + FP)}$$

召回率的计算方式为：

$$recall = \frac{TP}{(TP + FN)}$$

精确率与召回率分别衡量了模型的不同表现，精确率说的是，如果模型认为一张图片是 Logo 类，那有多大概率是 Logo 类。而召回率衡量的是，在整个验证集中，模型能找到多少 Logo 图片。

那问题来了，怎样根据这两个指标来选择模型呢？业务需求不同，我们侧重的指标就不一样。

比如在我们的这个项目中，如果老板允许一部分 Logo 图片没有被识别，但是模型必须非常准，模型说一张图片是 Logo 类，那图片真实类别就有非常大的概率是 Logo 类图片，那应该侧重的就是精确率；如果老板希望把线上 Logo 类尽可能地识别出来，允许一部分图片被误识别，那应该侧重的就是召回率。

在计算精确率与召回率的时候，给你分享一下我的经验。在实际项目中，我习惯把模型对每张图片的预测结果保存到一个 txt 中，这样可以比较直观地筛选一些模型的 badcase，并且验证集如果非常大，又需要调整的时候，直接更改 txt 就可以了，不需要再次让模型预测整个验证集。

下面是 txt 文件的一部分，分别记录了 logo 类的概率、others 类的概率、真实类别是否为 logo、真实类别是否为 others、预测类别是否为 logo、预测类别是否为 ohters、图片

名。

14.jpeg 是开篇例子的那张图片，模型认为它是 Logo 的概率是 0.58476，others 类的概率是 0.41524。

```
...
0.64460 0.35540 1 0 1 0 ./data/val/logo/13.jpeg
0.58476 0.41524 1 0 1 0 ./data/val/logo/14.jpeg
...
```

下图是我训练了 10 个 Epoch 的 B0 模型，在验证集 (这里我用训练集充当了一下验证集) 上的评价效果。

```
***** Classify Metric Test results *****

+--------+-----------+--------+
| Metric | precision | recall |
+--------+-----------+--------+
|  logo  |    1.0    |  0.8   |
| others |  0.83333  |  1.0   |
+--------+-----------+--------+

***** Multi-Class Confusion Matrix *****

+-----------+----------+------------+
| confusion | pre_logo | pre_others |
+-----------+----------+------------+
|    logo   |    8     |     2      |
|   others  |    0     |     10     |
+-----------+----------+------------+
```

通过混淆矩阵可以看到，整个验证集一共有 8+0 张图片被预测为 logo 类，所以 logo 类的精确率为 8 / (8 + 0) = 1；logo 类一共有 8+2 张图片，有两张预测错了，所以召回率为 8 / (8 +2) = 0.8。

others 类别的计算类似，你可以自己算算看。

小结

恭喜你，完成了今天的学习任务。今天我们一起完成了一个图像分类项目的实践。虽然项目规模较小，但是在真实项目中的每一个环节都包含在内了，可以说是麻雀虽小，五脏俱全。

下面我们回顾一下每个环节上的关键要点和实操经验。

数据准备其实是最关键的一步，数据的质量直接决定了模型好坏。所以，在开始训练之前你应该对你的数据集有十足的了解才可以。例如，验证集还是否可以反映出训练集、数据中有没有脏数据、数据分布有没有偏等等。

完成数据准备之后就到了模型训练，图像分类任务其实基本上都是采用主流的卷积神经网络了，很少对模型结构做一些更改。

最后的模型评估环节要侧重业务场景，看业务上需要高精确还是高召回，然后再对你的模型做调整。

思考题

老板希望你的模型能尽可能的把线上所有极客时间的海报都找到，允许一些误召回。训练模型的时候你应该侧重精确率还是召回率？

推荐你动手实现一下今天的 Demo，也欢迎你把这节课分享给更多的同事、朋友，跟他一起学习进步。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 17 | 图像分类（上）：图像分类原理与图像分类模型

下一篇 19 | 图像分割（上）：详解图像分割原理与图像分割模型

精选留言 (7)

写留言

clee
2021-11-29

你好，训练好的模型进行评估，从结果来看，好像都是都被分类为了Logo，可能哪儿出现了问题呢？以下是测试代码，另外有个问题是从tensorboard进行Loss数据分析发现，只展示最后一个epoch的Loss点？

```
model = EfficientNet.from_name('efficientnet-b0')
num_ftrs = model._fc.in_features…
```

展开

作者回复: 你好，代码看起来没有问题。可以看看数据分布是否平衡么、训练时是否使用pretrain model了以及模型是否收敛了。

Tensorboard中的loss你是说是一条直线吗？直线的数值等于最后一个epoch的loss？

narsil的梦

2021-11-24

老师好，请问下面这段代码里为什么调用 model 的时候一次传入了一个 batch 的图片数量而不是一张一张传入并计算 output？如果一次传入多张图片输入，是不是输出的 output 是平均值？但我看 EfficientNet 的 forward 函数确实一次只接受一张图片输入

for i, (images, target) in enumerate(train_loader):…

展开

作者回复: 你好，narsil的梦，感谢你的留言。

forward接受的也是一个batch的数据。

forward的输入即为网络的输入，它是一个shape为（batch_size, 通道数，高，宽）的tensor。

output的输出也是一个batch_size的输出，而不是平均值。

当预测一张图片的时候，输入应该为(1, 通道数，高，宽)的tensor。

共 2 条评论 >

vcjmhg

2021-11-24

思考题：

召回率衡量的是，在整个验证集中，模型能找到多少 Logo 图片。因此在尽可能的把线上所有极客时间的海报都找到，允许一些误召回的情况下，训练的的模型应该更侧重召回率。

展开

作者回复: 👍👍^^

Geek_8cfc4c

2021-11-23

您好：

训练完成后，我尝试加载checkpoint里面模型进行推断，代码如下

```

from efficientnet import EfficientNet…

展开

作者回复: 你好，感谢你的留言和认可。

model = EfficientNet.from_name( ) 之后需要修改一下全连接层的神经元的数目。
```

可以用下面的代码试试。
（应该是这个问题，如果不是的话，可以进微信群，把你的代码发给我看看，或者新开一个留言让我再看看提示什么信息。^^）

```
from efficientnet import EfficientNet
model = EfficientNet.from_name('efficientnet-b0')
num_ftrs = model._fc.in_features
model._fc = nn.Linear(num_ftrs, args.classes_num)

model.load_state_dict(torch.load('checkpoints&#47;checkpoint.pth.tar.epoch_19'))
```

王珅
2021-11-23

模型能尽可能的把线上所有极客时间的海报都找到，则应该侧重召回率，召回率越高越好。

作者回复: 👍👍^^

zhaobk
2021-11-22

老师好。可以说说练完以后，要这么进行验证吗？

展开

作者回复: 你好，zhaobk，谢谢留言。你说的验证是指如何评估吗？
评估的话可以按照文稿中的精确与召回对每个模型进行评估，然后选择合适的模型。
代码的话，sklearn中提供了混淆矩阵、精确和召回的方法。

Geek_a26208
2021-11-22

Epoch 18 tensor(0.5764, grad_fn=<NllLossBackward>)
Epoch 18 tensor(0.8147, grad_fn=<NllLossBackward>)
Epoch 19 tensor(0.3332, grad_fn=<NllLossBackward>)

Epoch 19 tensor(0.1257, grad_fn=<NllLossBackward>)
Epoch 19 tensor(0.2692, grad_fn=<NllLossBackward>)…
展开

作者回复: 你好，感谢你的留言。看起来似乎日志打印有点问题。一个epoch打印了多次。
如果没解决的话，需要提供再详细一点的信息。

共 2 条评论